CHAPTER 11

Basic I/O Interface

INTRODUCTION

A microprocessor is great at solving problems, but if it can't communicate with the outside world, it is of little worth. This chapter outlines some of the basic methods of communications, both serial and parallel, between humans or machines and the microprocessor.

In this chapter, we first introduce the basic I/O interface and discuss decoding for I/O devices. Then, we provide detail on parallel and serial interfacing, both of which have a variety of applications. To study applications, we connect analog-to-digital and digital-to-analog converters, as well as both DC and stepper motors to the microprocessor.

CHAPTER OBJECTIVES

Upon completion of this chapter, you will be able to:

1. Explain the operation of the basic input and output interfaces.
2. Decode an 8-, 16-, and 32-bit I/O device so that they can be used at any I/O port address.
3. Define handshaking and explain how to use it with I/O devices.
4. Interface and program the 82C55 programmable parallel interface.
5. Interface LCD displays, LED displays, keyboards, ADC, DAC, and various other devices to the 82C55.
6. Interface and program the 16550 serial communications interface adapter.
7. Interface and program the 8254 programmable interval timer.
8. Interface an analog-to-digital converter and a digital-to-analog converter to the microprocessor.
9. Interface both DC and stepper motors to the microprocessor.

11–1 INTRODUCTION TO I/O INTERFACE

In this section of the text I/O instructions (IN, INS, OUT, and OUTS) are explained and used in example applications. Also explained here is the concept of isolated (sometimes called direct or I/O mapped I/O) and memory-mapped I/O, the basic input and output interfaces, and handshaking. A working knowledge of these topics makes it easier to understand the connection and

operation of the programmable interface components and I/O techniques presented in the remainder of this chapter and text.

The I/O Instructions

The instruction set contains one type of instruction that transfers information to an I/O device (OUT) and another to read information from an I/O device (IN). Instructions (INS and OUTS, found on all versions except the 8086/8088) are also provided to transfer strings of data between the memory and an I/O device. Table 11–1 lists all versions of each instruction found in the microprocessor's instruction set.

Instructions that transfer data between an I/O device and the microprocessor's accumulator (AL, AX, or EAX) are called **IN** and **OUT**. The I/O address is stored in register DX as a 16-bit I/O address or in the byte (p8) immediately following the opcode as an 8-bit I/O address. Intel calls the 8-bit form (p8) a **fixed address** because it is stored with the instruction, usually in a ROM. The 16-bit I/O address in DX is called a **variable address** because it is stored in a DX, and then used to address the I/O device. Other instructions that use DX to address I/O are the INS and OUTS instructions. I/O ports are 8 bits in width so whenever a 16-bit port is accessed two consecutive 8-bit ports are actually addressed. A 32-bit I/O port is actually four 8-bit ports. For example, port 100H is accessed as a word, then 100H and 101H are actually accessed. Port 100H contains the least significant part of the data and port 101H the most significant part.

TABLE 11–1 Input/Output instructions.

Instruction	*Data Width*	*Function*
IN AL, p8	8	A byte is input into AL from port p8
IN AX, p8	16	A word is input into AX from port p8
IN EAX, p8	32	A doubleword is input into EAX from port p8
IN AL, DX	8	A byte is input into AL from the port addressed by DX
IN AX, DX	16	A word is input into AX from the port addressed by DX
IN EAX, DX	32	A doubleword is input into EAX from the port addressed by DX
INSB	8	A byte is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 1
INSW	16	A word is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 2
INSD	32	A doubleword is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 4
OUT p8, AL	8	A byte is output from AL into port p8
OUT p8, AX	16	A word is output from AL into port p8
OUT p8, EAX	32	A doubleword is output from EAX into port p8
OUT DX, AL	8	A byte is output from AL into the port addressed by DX
OUT DX, AX	16	A word is output from AX into the port addressed by DX
OUT DX, EAX	32	A doubleword is output from EAX into the port addressed by DX
OUTSB	8	A byte is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 1
OUTSW	16	A word is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 2
OUTSD	32	A doubleword is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 4

Whenever data are transferred by using the IN or OUT instructions, the I/O address, often called a **port number** (or simply port), appears on the address bus. The external I/O interface decodes the port number in the same manner that it decodes a memory address. The 8-bit fixed port number (p8) appears on address bus connections A_7–A_0 with bits A_{15}–A_8 equal to 00000000_2. The address connections above A_{15} are undefined for an I/O instruction. The 16-bit variable port number (DX) appears on address connections A_{15}–A_0. This means that the first 256 I/O port addresses (00H–FFH) are accessed by both the fixed and variable I/O instructions, but any I/O address from 0100H to FFFFH is only accessed by the variable I/O address. In many dedicated systems, only the rightmost 8 bits of the address are decoded, thus reducing the amount of circuitry required for decoding. In a PC computer, all 16 address bus bits are decoded with locations 0000H–03FFH, which are the I/O addresses used for I/O inside the PC on the ISA (**industry standard architecture**) bus.

The INS and OUTS instructions address an I/O device by using the DX register, but do not transfer data between the accumulator and the I/O device as do the IN and OUT instructions. Instead, these instructions transfer data between memory and the I/O device. The memory address is located by ES:DI for the INS instruction and by DS:SI for the OUTS instruction. As with other string instructions, the contents of the pointers are incremented or decremented, as dictated by the state of the direction flag (DF). Both INS and OUTS can be prefixed with the REP prefix, allowing more than one byte, word, or doubleword to be transferred between I/O and memory.

The Pentium 4 and Core2 operating in the 64-bit mode have the same I/O instructions. There are no 64-bit I/O instructions in the 64-bit mode. The main reason is that most I/O is still 8 bits and likely will remain so for an indefinite time.

Isolated and Memory-Mapped I/O

There are two different methods of interfacing I/O to the microprocessor: **isolated I/O** and **memory-mapped I/O**. In the isolated I/O scheme, the IN, INS, OUT, and OUTS instructions transfer data between the microprocessor's accumulator or memory and the I/O device. In the memory-mapped I/O scheme, any instruction that references memory can accomplish the transfer. Both isolated and memory-mapped I/O are in use, so both are discussed in this text. The PC does not use memory-mapped I/O.

Isolated I/O. The most common I/O transfer technique used in the Intel microprocessor-based system is isolated I/O. The term *isolated* describes how the I/O locations are isolated from the memory system in a separate I/O address space. (Figure 11–1 illustrates both the isolated and memory-mapped address spaces for any Intel 80X86 or Pentium–Core2 microprocessor.) The addresses for isolated I/O devices, called ports, are separate from the memory. Because the ports are separate, the user can expand the memory to its full size without using any of memory space for I/O devices. A disadvantage of isolated I/O is that the data transferred between I/O and the microprocessor must be accessed by the IN, INS, OUT, and OUTS instructions. Separate control signals for the I/O space are developed (using M/$\overline{\text{IO}}$ and W/$\overline{\text{R}}$), which indicate an I/O read ($\overline{\text{IORC}}$) or an I/O write ($\overline{\text{IOWC}}$) operation. These signals indicate that an I/O port address, which appears on the address bus, is used to select the I/O device. In the personal computer, isolated I/O ports are used for controlling peripheral devices. An 8-bit port address is used to access devices located on the system board, such as the timer and keyboard interface, while a 16-bit port is used to access serial and parallel ports as well as video and disk drive systems.

Memory-Mapped I/O. Unlike isolated I/O, memory-mapped I/O does not use the IN, INS, OUT, or OUTS instructions. Instead, it uses any instruction that transfers data between the microprocessor and memory. A memory-mapped I/O device is treated as a memory location in the memory map. The main advantage of memory-mapped I/O is that any memory transfer instruction can be used to access the I/O device. The main disadvantage is that a portion of the memory system is used as the I/O map. This reduces the amount of memory available to applications. Another advantage is that the $\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$ signals have no function in a memory-mapped I/O system and may reduce the amount of circuitry required for decoding.

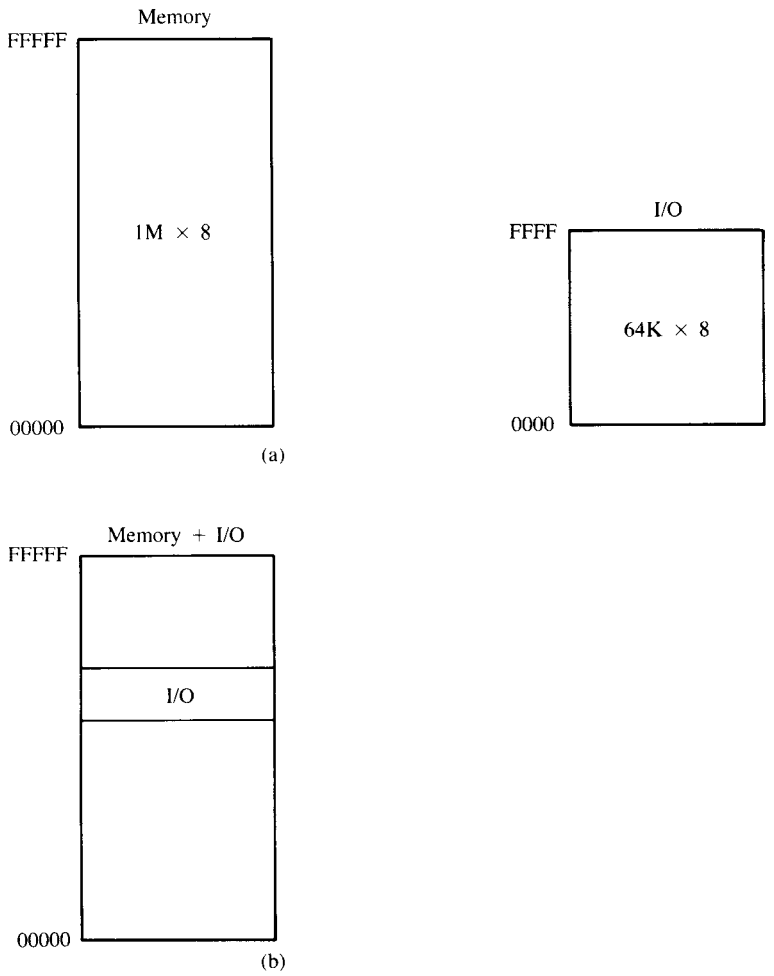


FIGURE 11–1 The memory and I/O maps for the 8086/8088 microprocessors. (a) Isolated I/O. (b) Memory-mapped I/O.

Personal Computer I/O Map

The personal computer uses part of the I/O map for dedicated functions. Figure 11–2 shows the I/O map for the PC. Note that I/O space between ports 0000H and 03FFH is normally reserved for the computer system and the ISA bus. The I/O ports located at 0400H–FFFFH are generally available for user applications, main-board functions, and the PCI bus. Note that the 80287 arithmetic coprocessor uses I/O address 00F8H–00FFH for communications. For this reason, Intel reserves I/O ports 00F0H–00FFH. The 80386–Core2 use I/O ports 800000F8–800000FFH for communications to their coprocessors. The I/O ports located between 0000H and 00FFH are accessed via the fixed port I/O instructions; the ports located above 00FFH are accessed via the variable I/O port instructions.

Basic Input and Output Interfaces

The basic input device is a set of three-state buffers. The basic output device is a set of data latches. The term IN refers to moving data from the I/O device into the microprocessor and the term OUT refers to moving data out of the microprocessor to the I/O device.

The Basic Input Interface. Three-state buffers are used to construct the 8-bit input port depicted in Figure 11–3. The external TTL data (simple toggle switches in this example) are connected to the

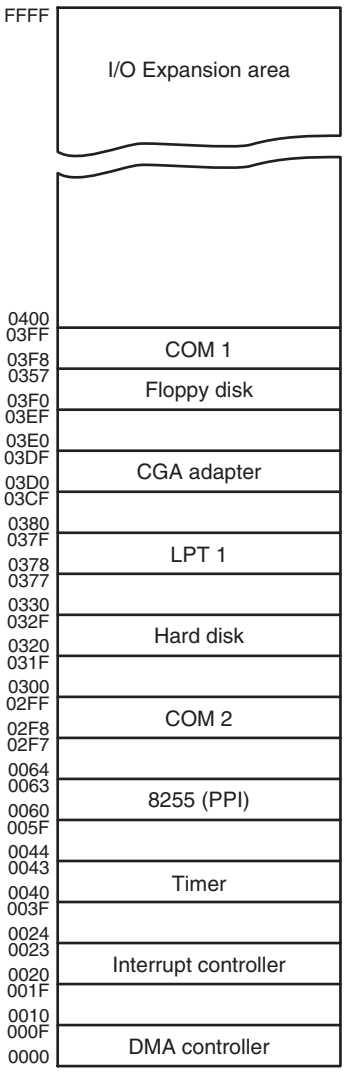


FIGURE 11–2 The I/O map of a personal computer illustrating many of the fixed I/O areas.

inputs of the buffers. The outputs of the buffers connect to the data bus. The exact data bus connections depend on the version of the microprocessor. For example, the 8088 has data bus connections D_7–D_0, the 80386/80486 has connections D_{31}–D_0, and the Pentium–Core2 have connections D_{63}–D_0. The circuit of Figure 11–3 allows the microprocessor to read the contents of the eight switches that connect to any 8-bit section of the data bus when the select signal $\overline{\text{SEL}}$ becomes a logic 0. Thus, whenever the IN instruction executes, the contents of the switches are copied into the AL register.

When the microprocessor executes an IN instruction, the I/O port address is decoded to generate the logic 0 on $\overline{\text{SEL}}$. A 0 placed on the output control inputs ($\overline{1\text{G}}$ and $\overline{2\text{G}}$) of the 74ALS244 buffer causes the data input connections (A) to be connected to the data output (Y) connections. If a logic 1 is placed on the output control inputs of the 74ALS244 buffer, the device enters the three-state high-impedance mode that effectively disconnects the switches from the data bus.

This basic input circuit is not optional and must appear any time that input data are interfaced to the microprocessor. Sometimes it appears as a discrete part of the circuit, as shown in Figure 11–3; many times it is built into a programmable I/O device.

Sixteen- or 32-bit data can also be interfaced to various versions of the microprocessor, but this is not nearly as common as using 8-bit data. To interface 16 bits of data, the circuit in

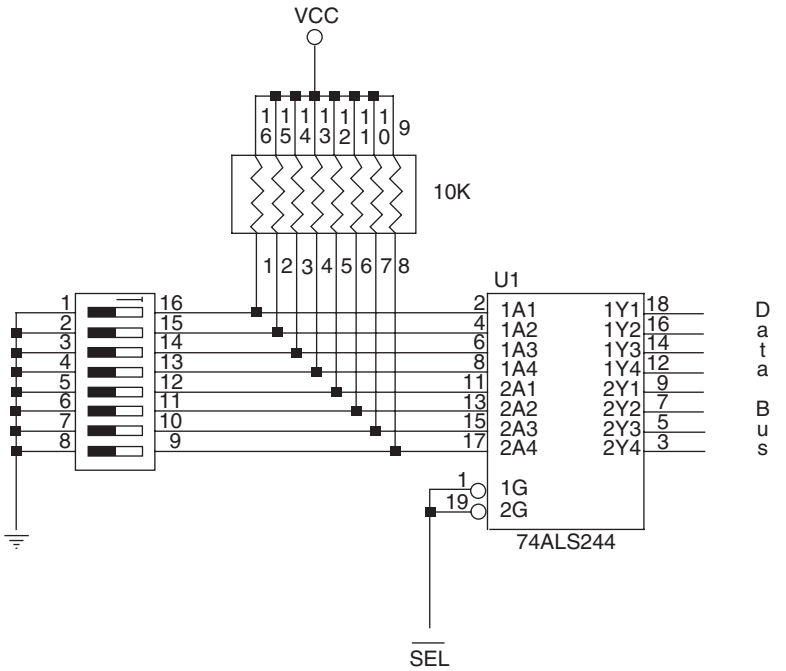


FIGURE 11–3 The basic input interface illustrating the connection of eight switches. Note that the 74ALS244 is a three-state buffer that controls the application of the switch data to the data bus.

Figure 11–3 is doubled to include two 74ALS244 buffers that connect 16 bits of input data to the 16-bit data bus. To interface 32 bits of data, the circuit is expanded by a factor of 4.

The Basic Output Interface. The basic output interface receives data from the microprocessor and usually must hold it for some external device. Its latches or flip-flops, like the buffers found in the input device, are often built into the I/O device.

Figure 11–4 shows how eight simple light-emitting diodes (LEDs) connect to the microprocessor through a set of eight data latches. The latch stores the number output by the microprocessor from the data bus so that the LEDs can be lit with any 8-bit binary number. Latches are needed to hold the data because when the microprocessor executes an OUT instruction, the data are only present on the data bus for less than 1.0 μs. Without a latch, the viewer would never see the LEDs illuminate.

When the OUT instruction executes, the data from AL, AX, or EAX are transferred to the latch via the data bus. Here, the D inputs of a 74ALS374 octal latch are connected to the data bus to capture the output data, and the Q outputs of the latch are attached to the LEDs. When a Q output becomes a logic 0, the LED lights. Each time that the OUT instruction executes, the $\overline{\text{SEL}}$ signal to the latch activates, capturing the data output to the latch from any 8-bit section of the data bus. The data are held until the next OUT instruction executes. Thus, whenever the output instruction is executed in this circuit, the data from the AL register appear on the LEDs.

Handshaking

Many I/O devices accept or release information at a much slower rate than the microprocessor. Another method of I/O control, called **handshaking** or **polling**, synchronizes the I/O device with the microprocessor. An example of a device that requires handshaking is a parallel printer that prints a few hundred characters per second (CPS). It is obvious that the microprocessor can send more than a few hundred CPS to the printer, so a way to slow the microprocessor down to match speeds with the printer must be developed.

Figure 11–5 illustrates the typical input and output connections found on a printer. Here, data are transferred through a series of data connections (D_7–D_0). BUSY indicates that the printer is busy. $\overline{\text{STB}}$ is a clock pulse used to send data to the printer for printing.

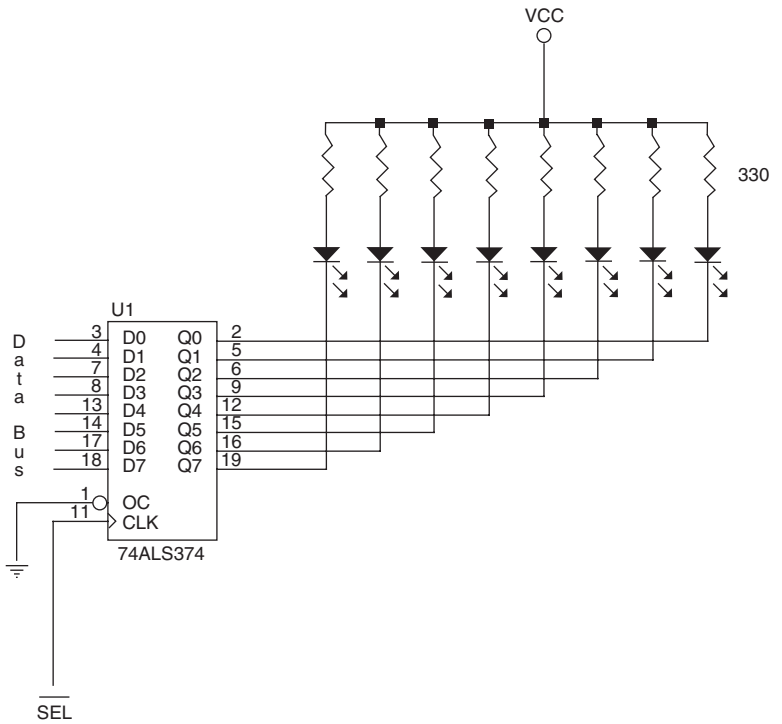


FIGURE 11–4 The basic output interface connected to a set of LED displays.

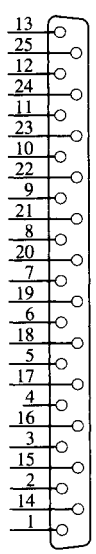
The ASCII data to be printed by the printer are placed on D_7–D_0, and a pulse is then applied to the $\overline{STB}$ connection. The strobe signal sends or clocks the data into the printer so that they can be printed. As soon as the printer receives the data, it places a logic 1 on the BUSY pin, indicating that the printer is busy printing data. The microprocessor software polls or tests the BUSY pin to decide whether the printer is busy. If the printer is busy, the microprocessor waits; if it is not busy, the microprocessor sends the next ASCII character to the printer. This process of interrogating the printer, or any asynchronous device like a printer, is called handshaking or polling. Example 11–1 illustrates a simple procedure that tests the printer BUSY flag and then sends data to the printer if it is not busy. Here, the PRINT procedure prints the ASCII-coded contents of BL only if the BUSY flag is a logic 0, indicating that the printer is not busy. This procedure is called each time a character is to be printed.

EXAMPLE 11–1

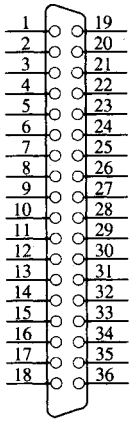
```
;An assembly language procedure that prints the ASCII contents of BL.

PRINT   PROC    NEAR

        .REPEAT                     ;test the busy flag
                IN AL,BUSY
                TEST AL,BUSY_BIT
        .UNTIL ZERO
        MOV AL,BL                   ;position data in AL
        OUT PRINTER,AL              ;print data
        RET

PRINT   ENDP
```

Notes about Interfacing Circuitry

A part of interfacing requires some knowledge about electronics. This portion of the introduction to interfacing examines some of the many facets of electronic interfacing. Before a circuit or



Connector DB25



Connector CENT36

DB25 Pin number	CENT36 Pin number	Function	DB25 Pin number	CENT36 Pin number	Function
1	1	$\overline{\text{Data Strobe}}$	12	12	Paper empty
2	2	Data 0 (D0)	13	13	Select
3	3	Data 1 (D1)	14	14	Afd
4	4	Data 2 (D2)	15	32	$\overline{\text{Error}}$
5	5	Data 3 (D3)	16	—	$\overline{\text{RESET}}$
6	6	Data 4 (D4)	17	31	Select in
7	7	Data 5 (D5)	18—25	19—30	Ground
8	8	Data 6 (D6)	—	17	Frame ground
9	9	Data 7 (D7)	—	16	Ground
10	10	$\overline{\text{Ack}}$	—	33	Ground
11	11	Busy			

FIGURE 11–5 The DB25 connector found on computers and the Centronics 36-pin connector found on printers for the Centronics parallel printer interface.

device can be interfaced to the microprocessor, the terminal characteristics of the microprocessor and its associated interfacing components must be known. (This subject was introduced at the start of Chapter 9.)

Input Devices. Input devices are already TTL and compatible, and therefore can be connected to the microprocessor and its interfacing components, or they are switch-based. Most switch-based devices are either open or connected. These are not TTL levels—TTL levels are a logic 0 (0.0 V–0.8 V) or a logic 1 (2.0 V–5.0 V).

For a switch-based device to be used as a TTL-compatible input device, some conditioning must be applied. Figure 11–6 shows a simple toggle switch that is properly connected to function as an input device. Notice that a pull-up resistor is used to ensure that when the switch is open, the output signal is a logic 1; when the switch is closed, it connects to ground, producing a valid logic 0 level. The value of the pull-up resistor is not critical—it merely assures that the signal is

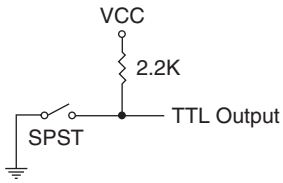


FIGURE 11–6 A single-pole, single-throw switch interfaced as a TTL device.

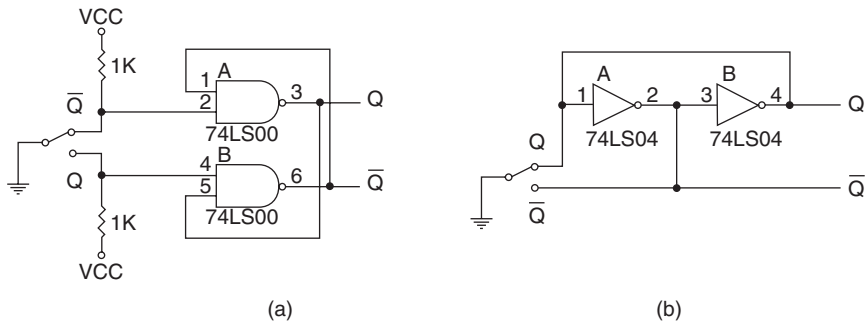


FIGURE 11–7 Debouncing switch contacts: (a) conventional debouncing and (b) practical debouncing.

at a logic 1 level. A standard range of values for pull-up resistors is usually anywhere between 1K Ω and 10K Ω.

Mechanical switch contacts physically bounce when they are closed, which can create a problem if a switch is used as a clocking signal for a digital circuit. To prevent problems with bounces, one of the two circuits depicted in Figure 11–7 can be constructed. The first circuit (a) is a classical textbook bounce eliminator; the second (b) is a more practical version of the same circuit. Because the first version costs more money to construct, in practice, the second would be used because it requires no pull-up resistors and only two inverters instead of two NAND gates.

You may notice that both circuits in Figure 11–7 are asynchronous flip-flops. The circuit of (b) functions in the following manner: Suppose that the switch is currently at position $\overline{Q}$. If it is moved toward Q but does not yet touch Q, the Q output of the circuit is a logic 0. The logic 0 state is remembered by the inverters. The output of inverter B connects to the input of inverter A. Because the output of inverter B is a logic 0, the output of inverter A is a logic 1. The logic 1 output of inverter A maintains the logic 0 output of inverter B. The flip-flop remains in this state until the moving switch-contact first touches the Q connection. As soon as the Q input from the switch becomes a logic 0, it changes the state of the flip-flop. If the contact bounces back away from the Q input, the flip-flop remembers and no change occurs, thus eliminating any bounce.

Output Devices. Output devices are far more diverse than input devices, but many are interfaced in a uniform manner. Before any output device can be interfaced, we must understand what the voltages and currents are from the microprocessor or a TTL interface component. The voltages are TTL-compatible from the microprocessor of the interfacing element. (Logic 0 = 0.0 V to 0.4 V; logic 1 = 2.4 V to 5.0 V.) The currents for a microprocessor and many microprocessor-interfacing components are less than for standard TTL components. (Logic 0 = 0.0 to 2.0 mA; logic 1 = 0.0 to 400 μA.)

Once the output currents are known, a device can now be interfaced to one of the outputs. Figure 11–8 shows how to interface a simple LED to a microprocessor peripheral pin. Notice that a transistor driver is used in Figure 11–8(a) and a TTL inverter is used in Figure 11–8(b). The TTL inverter (standard version) provides up to 16 mA of current at a logic 0 level, which is more than enough to drive a standard LED. A standard LED requires 10 mA of forward bias current to light. In both circuits, we assume that the voltage drop across the LED is about 2.0 V.

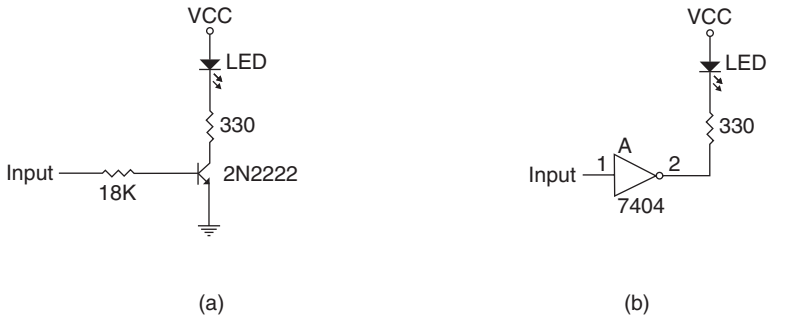


FIGURE 11–8 Interfacing an LED: (a) using a transistor and (b) using an inverter.

The data sheet for an LED states that the nominal drop is 1.65 V, but it is known from experience that the drop is anywhere between 1.5 V and 2.0 V. This means that the value of the current-limiting resistor is 3.0 V ÷ 10 mA or 300 Ω. Because 300 Ω is not a standard resistor value (the lowest cost), a 330 Ω resistor is chosen for this interface.

In the circuit of Figure 11–8(a), we elected to use a switching transistor in place of the TTL buffer. The 2N2222 is a good low-cost, general-purpose switching transistor that has a minimum gain of 100. In this circuit, the collector current is 10 mA, so the base current will be 1/100 of the collector current of 0.1 mA. To determine the value of the base current–limiting resistor, use the 0.1 mA base current and a voltage drop of 1.7 V across the base current–limiting resistor. The TTL input signal has a minimum value of 2.4 V and the drop across the emitter-base junction is 0.7 V. The difference is 1.7 V, which is the voltage drop across the resistor. The value of the resistor is 1.7 V ÷ 0.1 mA or 17K Ω. Because 17K Ω is not a standard value, an 18K Ω resistor is chosen.

Suppose that we need to interface a 12 V DC motor to the microprocessor and the motor current is 1A. Obviously, we cannot use a TTL inverter for two reasons: The 12 V signal would burn out the inverter and the amount of current far exceeds the 16 mA maximum current from the inverter. We cannot use a 2N2222 transistor either, because the maximum amount of current is 250 mA to 500 mA, depending on the package style chosen. The solution is to use a Darlington-pair, such as a TIP120. The TIP120 costs 25¢ and with the proper heat sink can handle 4A of current.

Figure 11–9 illustrates a motor connected to the Darlington-pair. The Darlington-pair has a minimum current gain of 7000 and a maximum current of 4A. The value of the bias resistor is calculated exactly the same as the one used in the LED driver. The current through the resistor is 1.0 A ÷ 7000, or about 0.143 mA. The voltage drop across the resistor is 0.9 V because of the two diode drops (base/emitter junctions) instead of one. The value of the bias resistor is 0.9 V ÷ 0.143 mA or 6.29K Ω. The standard value of 6.2 K Ω is used in the circuit. The Darlington-pair must use a heat sink because of the amount of current going through it. Typically any device that passes more than ½ A of current needs a heat sink. The diode must also be present to prevent the Darlington-pair from being destroyed by the inductive kickback from the motor. This circuit is also used to interface mechanical relays or just about any device that requires a large amount of current or a change in voltage.

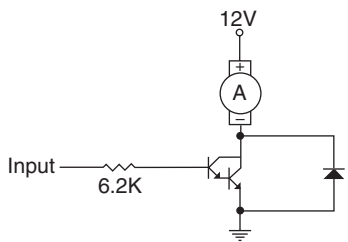


FIGURE 11–9 A DC motor interfaced to a system by using a Darlington-pair.